Yasmin ayada



كلية هندسة المعلوماتية

السنة الثانية

2024-2025

داخلة

داخل كل شي لعند محاضرة 7 باستثناء المحاضرة 8



خوارزميات وبنى المعطيات 1

القسم العملي

مدرس المقرر

م.باسل دهيمش

مقرر كامل







مراجعة تعليمات البرمجة

سنقوم من خلال التمرين التالي بمراجعة بعض تعليمات لغة C++.

تمرين: اكتب برنامج لاستصدار فواتير الكهرباء وفق شرائح الاستهلاك التالية:

يحتسب الاستهلاك من 1-600 كيلواط ساعي بكلفة 10 ليرات لكل كيلواط ساعي.

يحتسب الاستهلاك من 601-1000 كيلواط ساعي بكلفة 25 ليرة لكل كيلواط ساعي.

يحتسب الاستهلاك من 1001 كيلواط ساعي فما فوق بكلفة 135 ليرة لكل كيلواط ساعي.

تحتسب ضريبة بمقدار 20% من اجمالي الفاتورة، والمطلوب: أوجد قيمة فاتورة الكهرباء حسب قراءة مقطوعية العداد التي سيتم ادخالها.

الحل:

```
#include <iostream>
using namespace std;

void main ()
{
int x,y,z,a,sum=0,t, bill=0;
cin>>a;
if (a<=600) {x=a; y=0; z=0;}
if (a>600 && a<=1000) { x=600; y=a-600; z=0;}
if (a>1000) {x=600; y=400; z=a-1000;}
sum=x*10+y*25+z*135;
t=sum*20/100;
bill=sum+t;
cout<<bill;
system ("pause");
}
```

تدريب: قم بتعديل التمرين السابق باستخدام التعليمات if else و else، وناقش الحل.

تدريب: قم بتعديل التمرين السابق لتكرار عملية الإدخال وحساب الفواتير، علماً أننا نرغب بإيقاف عمليات الحساب والفوترة عند ادخال استهلاك قيمته -1.

الحل:

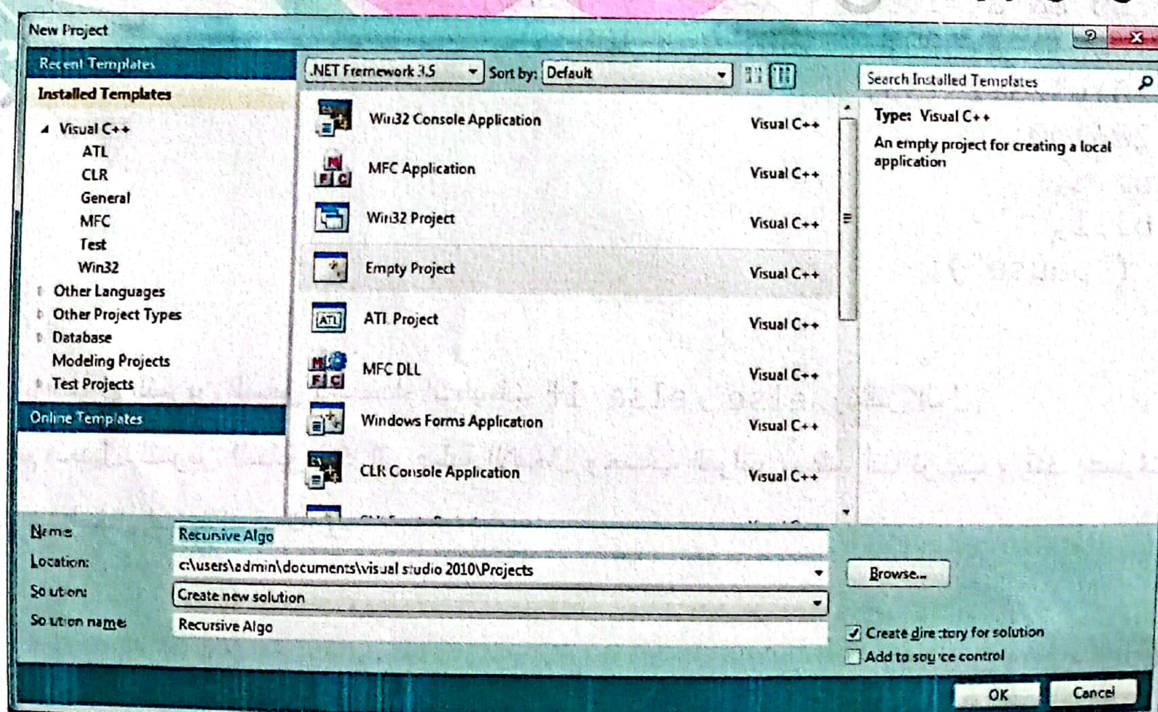
```
#include <iostream>
using namespace std;
```

```
void main ()
{
int x,y,z,a,sum=0,t, bill=0;
cin>>a;
while (a !=-1)
{
if (a>=0 && a<=600) {x=a; y=0; z=0;}
if (a>600 && a<=1000) { x=600; y=a-600; z=0;}
if (a>1000) {x=600; y=400; z=a-1000;}
sum=x*10+y*25+z*135;
t=sum*20/100;
bill=sum+t;
cout<<bill<<endl;
cin>>a;
}
system ("pause");
}
```

تذكير بكيفية التعامل مع Visual Studio

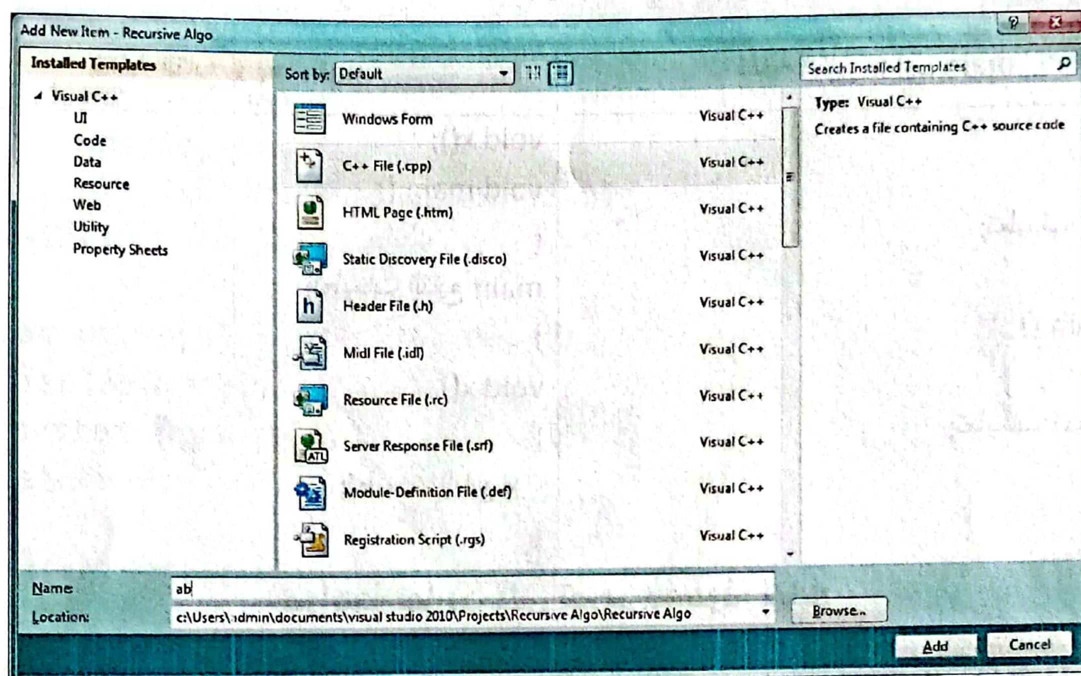
يمكن تنفيذ تعليمات لغة ++C للخوارزميات المعطاة بهذا المنهاج بأي مترجم، كمترجم Visual Studio أو ++DevC، وسنوضح الآن كيفية التعامل مع Visual Studio:

ننشئ مشروع فارغ Empty، ونسميه كما يوضح ذلك الشكل التالي:

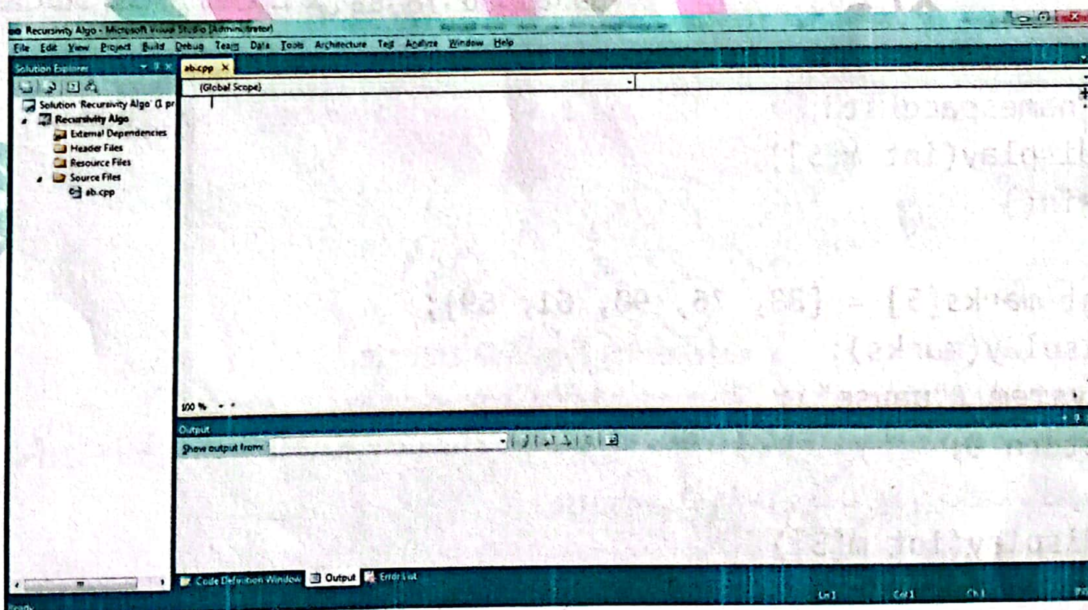


ثم من لوحة Solution Explorer ننقر بالزر الأيمن على اسم المشروع ونختار Add ثم New Item ونختار C++ File(.cpp) كما هو موضح بالشكل التالي:

نتابع ونقوم بتسمية الملف ضمن المشروع وفق الشكل التالي:



وعند الانتهاء من الخطوات السابقة ستكون واجهة كتابة الأكواد جاهزة وفق ما يلي:



تذكير بأماكن التصريح عن التوابع

تكتب التوابع قبل main كما العادة في كل التمارين، ولكن يمكننا كتابة التوابع بعد main وفق المقارنة التالية:

كتابة التابع x بعد main	كتابة التابع x قبل main
void x(); void main () { main تعليمات التابع; } void x() { x تعليمات التابع; }	void x() { x تعليمات التابع; } void main () { main تعليمات التابع; }

مراجعة تمارين على تمرير مصفوفة لتابع

مثال1: اكتب برنامج لطباعة علامات خمس طلاب مخزنة ضمن مصفوفة، على ان يتم تمرير المصفوفة لتابع يقوم بالطباعة، علماً أن العلامات هي 88، 76، 90، 61، 69

```
#include <iostream>
using namespace std;
void display(int m[5]);
int main()
{
    int marks[5] = {88, 76, 90, 61, 69};
    display(marks);
     system ("pause");
    return 0;
}
void display(int m[5])
{
    cout << "Displaying marks: "<< endl;

    for (int i = 0; i < 5; ++i)
    {
        cout << "Student "<< i + 1 <<": "<< m[i] << endl;
```

```
    }
}
```

مثال 2: قم بتعديل المثال السابق لمعالجة أول أربعة عناصر من المصفوفة وليس كل عناصرها الخمسة

الحل: هنا سنقوم بتمرير وسيطين للتابع display: الوسيط الأول هو المصفوفة، والوسيط الثاني هو عدد العناصر المراد معالجته في التابع.

```
#include <iostream>
using namespace std;
void display(int marks[5], int size);
int main()
{
    int marks[5] = {88, 76, 90, 61, 69};
    display(marks,4);
     system ("pause");
    return 0;
}
void display(int m[5], int size)
{
    cout << "Displaying marks: "<< endl;

    for (int i = 0; i < size; ++i)
    {
        cout << "Student "<< i + 1 <<": "<< m[i] << endl;
    }
}
```

مراجعة عن السجلات structs

تقوم المصفوفات بتخزين عناصر تنتمي إلى نفس نوع البيانات، لكن واقعياً في عالم البرمجة قد نحتاج إلى عناصر لها أنواع بيانات مختلفة، وهو ما تقوم به السجلات struct.

ملاحظة: تسمى حقول السجل بالأعضاء members.

ملاحظة: يسمى المتحول من نوع سجل بالكائن أو بالغرض (Object) أو بالنسخة (instance)

يوضح المثل التالي عملية التصريح عن السجل StudentRec الذي يخزن معلومات عن الطلاب والمتكون من الأعضاء name و idNum و percent، ثم انشاء غرض من هذا السجل

```
struct StudentRec
```

```
{
string name;
int idNum;
float percent;
};
StudentRec theStudent;
```

نتخيل أن الناتج عن التعليمة السابقة StudentRec theStudent; هو الشكل التالي، حيث يمثّل المتحول theStudent نسخة من السجل.

theStudent

theStudent
name
idNum
percent

ملاحظة: للوصول إلى أعضاء السجل (متحولات – توابع) باستخدام الكائن نكتب:

Objet name. Member name

مثال: اكتب برنامج يقوم بالتصريح عن سجل يحوي على الأعضاء: الاسم والكنية والعمر. قم بإنشاء غرض "نسخة" منه و أدخل البيانات في الغرض ثم اطبعها.

```
#include <iostream>
#include <string>
using namespace std;
struct PersonRec
{
string firstName;
string lastName;
int age;
};
void main()
{
PersonRec Person1;
cout << "Enter first name: ";
cin >> Person1.firstName;
```

```
cout << "Enter last name: ";
cin >> Person1.lastName;
cout << "Enter age: ";
cin >> Person1.age;
cout << "\n\nHello " << Person1.firstName << ' '<<
Person1.lastName << ". How are you?\n";
cout << "\n Your age is "<< Person1.age << ".\n";
system ("pause");
}
```

ملاحظة: يمكن التصريح عن مصفوفة عناصرها سجلات.

مراجعة عن الصفوف Classes

تتشابه كثيراً الصفوف مع السجلات في المفاهيم والتصريحات، مع بعض الاختلافات في الحالة الافتراضية لمحددات الوصول في كل منهما. يتخصص مقررين هما البرمجة الموجهة للكائنات1+2 بالصفوف ويعتمدان عليها في كتابة البرامج.

تنفيذ الخوارزميات العودية

وهي خوارزميات تستخدم ضمن تعليماتها استدعاء للخوارزمية نفسها. عند التعامل مع الخوارزميات العودية يجب أن تضمن انتهاء المعالجة بعد حد معين.

تستخدم بعض خوارزميات الترتيب الفكرة العودية كخوارزمية البحث الثنائي (Binary Search) وخوارزمية الترتيب السريع (Quick Sort) كما سنرى لاحقاً.

مثال1: اكتب برنامج (Code) خوارزمية حساب العاملي لعدد بالطريقة العودية:

الحل: شرط التوقف: fact(0)=1

```
#include <iostream>
using namespace std;
float fact (int n)
{ if (n==0) return 1;
else return (n * fact(n-1));
}

void main ()
{
    int n;
    cin>> n;
    cout<< "Factorial of n is "<< fact (n)<< endl;
    system ("pause");
}
```

مثال2: اكتب برنامج (Code) خوارزمية حساب سلسلة فيبوناشي :

الحل: شرط التوقف هو: Fib(0)=0 و Fib(1)=1

يتم حساب أي حد بجمع الحدين الذين قبله كالتالي:

Fib(n)= Fib(n-1) + Fib(n-2)

```
#include <iostream>
using namespace std;
int fib (int n)
{ if (n==0 || n==1) return n;
```

```
else return (fib(n-1)+fib(n-2));
}
void main ()
{
    int n;
    cin>> n;
    cout<< "The result is "<< fib (n)<< endl;
    system ("pause");
}
```

مثال3: اكتب كود خوارزمية أبراج هانوي ضمن الاستدعاءات العودية

الحل: شرط التوقف عندما يكون عدد الاقراص هو 1، عندها يلزم نقل مباشر للقرص من عمود المصدر A إلى عمود الهدف C.

بارامتر العمود المصدر — بارامتر العمود المساعد — بارامتر العمود الهدف

```
#include <iostream>
using namespace std;

void towerOfHanoi(int n, char A, char B, char C)
{
    if (n == 1)
    {
        cout << "Move disk 1 from rod " << A <<
                                 " to rod " << C<<endl;
        return;
    }
    towerOfHanoi(n - 1, A, C, B);
    cout << "Move disk " << n << " from rod " << A <<
                                 " to rod " << C << endl;
    towerOfHanoi(n - 1, B, A, C);
}


// Driver code
int main()
{
    int n = 3; // Number of disks
    towerOfHanoi(n, 'A', 'B', 'C'); // A, B and C are names of rods
    system("pause");
    return 0;
}
```

تدريب: أعد كتابة كود خوارزمية هانوي بحيث يكون تسلسل الوسطاء في التابع towerOfHanoi كالتالي:




```
void towerOfHanoi(int n, char A, char C, char B)
```

تدريب: كم عدد مرات تحريك n قرص لنقلهم من العمود المصدر إلى العمود الهدف حسب مسألة أبراج هانوي.

عدد الأقراص → عدد المرات التحريك

1 → 1
2 → 3
3 → 7
4 → 15

$2^n - 1$ الف نون



تنفيذ خوارزمية البحث الخطي (Linear Search) بالطريقة التكرارية

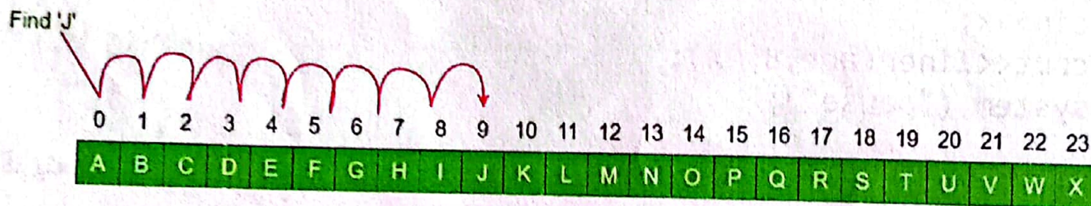
المشكلة: بفرض وجود مصفوفة هي []arr ولها عدد عناصر هي n، والمطلوب: اكتب برنامج يحوي على تابع للبحث الخطي عن عنصر x في المصفوفة :[]arr

طريقة البحث:

ابدأ من العنصر الأول في المصفوفة وقارنه مع العدد x،

اذا تم ايجاد العنصر في المصفوفة: فقم بالعودة برقم دليل[1] (index) العنصر وانهاء البحث.

اذا لم يتم ايجاد العنصر في المصفوفة: فقم بالعودة بالرقم 1-



مثال1: لتكن لدينا المصفوفة التالية []arr

arr[] = {44, 20, 80, 30, 60, 50, 110, 100, 130, 170}

ويراد البحث عن العنصر x = 110، أوجد دليل المصفوفة التي يحوي العنصر؟

الحل: الدليل هو 6 وبالتالي الخرج هو 6

مثال2: لتكن لدينا المصفوفة التالية []arr

arr[] = {44, 20, 80, 30, 60, 50, 110, 100, 130, 170}

ويراد البحث عن العنصر x = 175، أوجد دليل المصفوفة التي يحوي العنصر؟

الحل: العنصر غير موجود في المصفوفة وبالتالي الخرج هو 1-

[1] يسمى الدليل أيضاً بالفهرس

مثال1: اكتب برنامج (Code) خوارزمية البحث الخطي بلغة ++C:

```
#include <iostream>
using namespace std;
int liner(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}

void main()
{
     int arr[]={80,8,99,60,12,22,0,50};
     int x;
     cout<<"enter the element";
     cin>>x;
     cout<<liner(arr,8, x);
     system ("pause");
}
```

مثال2: قم بتعديل التابع liner السابق لحساب عدد مرات ظهور العنصر x

```
int liner(int arr[], int n, int x)
{
    int i,count=0;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
        count++;
    if (count ==0) return -1; else return count;
}
```

تنفيذ خوارزمية البحث الثنائي (Binary Search) بالطريقة العودية

يجب أن تكون المصفوفة مرتبة تصاعدياً – أو حتى تنازلياً – وفي حال لم تكن المصفوفة مرتبة تصاعدياً يجب علينا أولاً ترتيبها حتى نستطيع تطبيق هذه الخوارزمية.

طريقة البحث:

1- تبدأ الخوارزمية عملها بوضع مؤشر الفهرس في وسط المصفوفة حيث نقسم المصفوفة إلى قسمين.

2- يتم اختيار وسط المصفوفة من خلال المعادلة:

mid=(low+high)/2

3- تتم مقارنة العنصر الأوسط بالعنصر المراد البحث عنه والنتيجة إحدى 3 حالات:

- أن تتساوى قيمة العنصر المراد البحث عنه مع قيمة العنصر الأوسط وهنا نعود بقيمة الدليل وينهي البحث.
- أن تكون قيمة العنصر المراد البحث عنه أكبر من قيمة العنصر الأوسط، وبالتالي سيصبح الجزء الأيمن هو مدخلاً جديداً للخوارزمية.
- أن تكون قيمة العنصر المراد البحث عنه أصغر من قيمة العنصر الأوسط، وبالتالي سيصبح الجزء الأيسر هو مدخلاً جديداً للخوارزمية.

تتكرر الخطوات الثلاثة في النصف المتبقي حتى يتم ايجاد العنصر أو عدم ايجاده.

مثال1: اكتب برنامج (Code) خوارزمية البحث الثنائي بلغة C++:

```
#include <iostream>
using namespace std;
int binary(int arr[], int x, int low, int high)
{
int mid;
while(low<=high)
     {
     mid=(low+high)/2;
     if (x==arr[mid]) return mid;
     else if (x<arr[mid]) return binary(arr, x,low, mid-1);
     else return binary(arr, x,mid+1, high);
     }
return -1;
```

```
}
void main()
{
     int arr[]={20,30,50,60,90,100};
     int x;
     cin>>x;
     cout<<binary(arr, x,0,5);
     system ("pause");
}
```

تدريب: قم بتعديل التابع الرئيسي main ليتم إدخال قيم عناصر المصفوفة بواسطة المستخدم، وليس ضمن التصريح عن المصفوفة.



تنفيذ خوارزمية الترتيب بالاختيار (Selection Sort)

لترتيب المصفوفة تصاعديا حسب هذه الخوارزمية يتم التالي:

- تجزئتها إلى مصفوفتين: مصفوفة جزئية تم ترتيبها، ومصفوفة متبقية غير مرتبة
- في كل عملية بحث في المصفوفة غير المرتبة يتم ايجاد أصغر عنصر فيها ونقله إلى بدايتها ثم ضمه للجزء المرتب.
- نفرض في بداية كل شوط بحث بأن المتحول min يحتفظ بالعنصر الأصغر من المصفوفة

```
arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

اكتب برنامج (Code) خوارزمية الترتيب بالاختيار selectsort بلغة ++C:

```
#include <iostream>
using namespace std;

void selectsort (int arr[], int n)
{
     int i,j,min, temp;
     for (i=0; i<n-1; i++)
     {    min=i;
     for (j=i+1; j<n;j++)
```

```
if (arr[j]<arr[min])
     min=j;
temp=arr[i];
arr[i]= arr [min];
arr[min]=temp;
}
for (i=0;i<n; i++)
     cout <<arr[i]<<"  ";
}
     int main()
{
  int arr[] = {64, 25, 12, 22, 11};

  selectsort(arr, 5);
  system ("pause");
  return 0;
}
```



تنفيذ خوارزمية الترتيب الفقاعي (Bubble Sort)

يعتمد مبدأ عمل خوارزمية الترتيب الفقاعي على التبديل المتكرر بين العناصر المتجاورة إذا كانت هذه العناصر بترتيب خاطئ فيما بينها.

تبدأ المقارنات من اليسار بين العناصر المتجاورة، ويتم وضع العنصر الأكبر في أقصى اليمين.

تستمر أشواط التنفيذ ليتم توضيع ثاني أكبر عنصر في مكانه الصحيح... وهكذا

ملاحظة: كما العادة في مثل هذا النوع من الخوارزميات، يوجد حلقتان تكراريتان:

- الحلقة الخارجية مسؤولة عن عد الأشواط،
- والحلقة الداخلية مسؤولة عن اجراء عمليات المقارنة وتنفيذ مبدأ الخوارزمية.

اكتب برنامج (Code) خوارزمية الترتيب الفقاعي (Bubble Sort) بلغة ++C:

```
#include <iostream>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1])
              swap(arr[j], arr[j+1]);

        for (int i=0; i < n; i++)
       cout<< arr[i]<<"   ";
}
```

```
int main()
{
    int arr[] = {64, 34, 25, 12, 8, 11};
    bubbleSort(arr, 6);
     system ("pause");
    return 0;
}
```

تدريب: قم بتعديل الكود السابق ليلائم ترتيب عناصر المصفوفة تنازلياً.

تدريب: قم بتعديل الكود السابق لتنفيذ عميات التبديل يدوياً باستخدام متحول إضافي

تدريب: هل بالإمكان معرفة قانون عدد المقارنات الكلي اللازم لترتيب n عنصر حسب خوارزمية الترتيب الفقاعي؟

تدريب: تدريب: قم بتعديل برنامج الترتيب الفقاعي ليتم استدعائه بالتعليمة ;bubbleSort(arr, n) بدلاً من التعليمة ;bubbleSort(arr, 6)، مستخدماً الأمر sizeof() على أن يتم التعديل في بيئة Visual Studio حصراً.



ملاحظة: لاستكمال فهم مبدأ عمل أي خوارزمية يرجى مراجعة المحاضرات النظرية.

تنفيذ خوارزمية الترتيب بالحشر (Insertion Sort)

تعتمد فكرة هذه الخوارزمية على طريقة تشبه ترتيب ورق اللعب.

يتم في البداية تقسيم المصفوفة المعطاة إلى قسمين: قسم غير مرتب يميني، وقسم مرتب يساري، وفي بداية التنفيذ يحوي القسم المرتب على عنصر واحد فقط هو العنصر الاول في المصفوفة، وفي كل شوط يؤخذ عنصر من المصفوفة غير المرتبة ويضاف إلى مكانه الصحيح في المصفوفة المرتبة.

ملاحظة: يقف الدليل i على العنصر غير المرتب المراد حشره في المكان المناسب في المصفوفة المرتبة.

اكتب برنامج (Code) خوارزمية الترتيب بالحشر insertsort بلغة C++:

```
#include <iostream>
using namespace std;
void insertionsort (int arr[], int n)
{
    int i, temp, j;
    for (i = 1; i < n; i++)
    {
        temp = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > temp)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = temp;
    }
     for (i=0; i<n; i++)
     cout<<arr[i]<<"  ";
}
int main()
{
```

```
    int arr[] = {15, 25, 12, 22, 11};
    insertionsort(arr, 5);
     system ("pause");
    return 0;
}
```



هوداقلة

تنفيذ خوارزمية الترتيب السريع (Quick Sort)

يعتمد مبدأ هذه الخوارزمية على اختيار عنصر في المصفوفة يسمى محور(pivot) في أي مكان ضمنها وليكن آخرها مثلاً، ثم البحث عن مكانه المناسب ضمن المصفوفة ليقوم بتقسيمها إلى قسمين:

- القسم اليميني يجب أن تكون جميع عناصره أكبر من pivot،
- القسم اليساري يجب أن تكون جميع عناصره أصغر من pivot،

ثم تستدعى الخوارزمية استدعاءاً عودياً لكل قسم، وهكذا...

يحتفظ الدليل i بمكان العنصر الأصغر أو يساوي pivot مباشرة، أي أن العنصر الذي يليه هو أكبر من pivot.

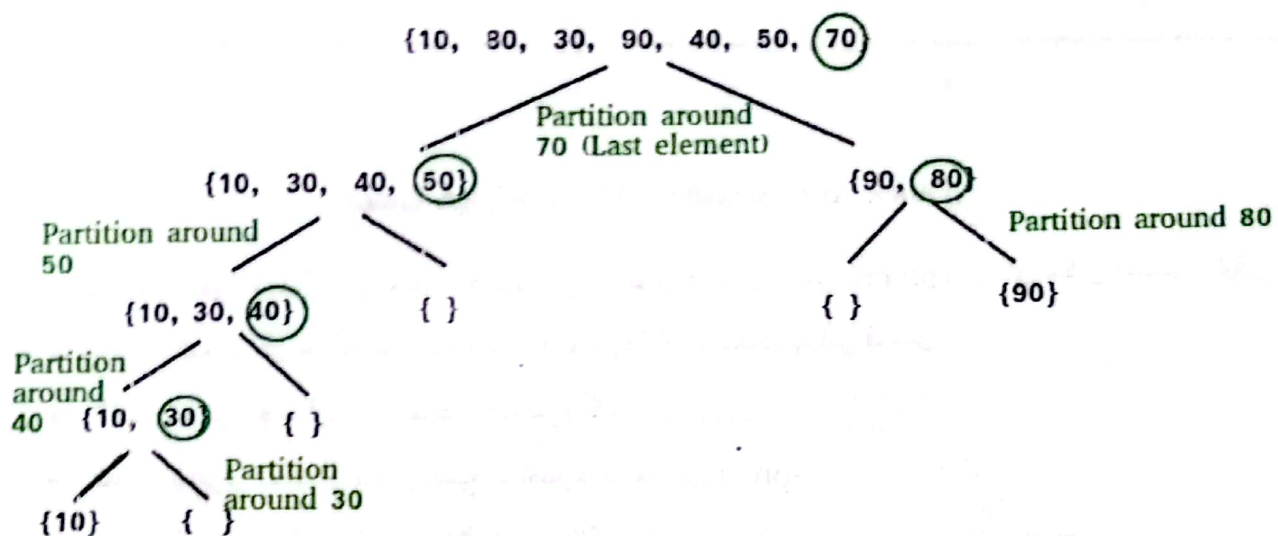
أثناء التجوال على المصفوفة: إذا وجدنا عبر الدليل j عنصر أقل من pivot فسنبدله مع العنصر الذي يلي i وهو أكبر من pivot كما أشرنا سابقا.

وكخلاصة: إن i+1 يشير إلى عنصر أكبر من pivot، و j يشير إلى عنصر أصغر من pivot، لذا يجب التبديل بينهما لجعل العناصر الأكبر من pivot تتحرك إلى يمينه، وجعل العناصر الأصغر منه تتحرك إلى يساره.

يقوم التابع `quickSort` بتأمين الاستدعاء العودي للمصفوفتين الجزئيتين الناتجتين عن تقسيم المصفوفة الرئيسية بواسطة pivot،

بينما يقوم التابع partition بإيجاد المكان المناسب للـpivot ضمن المصفوفة واجراء عمليات تبديل لتحسين ترتيبها وجعل العناصر الأصغر من pivot في القسم اليساري، وجعل العناصر الأكبر من pivot في القسم اليميني.

في الشكل التالي لدينا المصفوفة {10, 80, 30, 90, 40, 50, 70} واخترنا pivot هو آخر عنصر:



اكتب برنامج (Code) خوارزمية الترتيب السريع quicksort بلغة C++:

```
#include <iostream>
using namespace std;

int partition (int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element
  int temp;
    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++; // increment index of smaller element
               temp=arr[j];
               arr[j]=arr[i];
               arr[i]=temp;
        }
    }
              temp=arr[i+1];
              arr[i+1]=arr[high];
              arr[high]=temp;
    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
```

```
            int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void main()
{
    int arr[] = {10, 80, 30, 90, 40, 50, 70};
    quickSort(arr, 0, 6);
    cout << "Sorted array: \n";
    for (int i = 0; i <= 6; i++)
        cout << arr[i] << " ";
     system ("pause");
}
```



دائمة

مراجعة المؤشرات

ملاحظة: عند التصريح عن متحول بطريقة عادية لا يهمنا عنوان الموقع الذاكري لهذا المتحول ولحسن الحظ ان نظام التشغيل يحدده آلياً أثناء تنفيذ البرنامج حتى يبعد التعقيد عن المبرمج.

لكننا نحتاج أحياناً إلى التعامل مع مكونات الحاسوب الفيزيائية – كالذاكرة في مثالنا – بشكل مباشر والوصول إلى عناوين محددة فيها.

ملاحظة: يمكن معرفة عنوان المتحول والتعامل مع عنوانه عن طريق مفهوم المؤشر (Pointer)

ملاحظة: المؤشرات هي متحولات تخزن عناوين متحولات أخرى

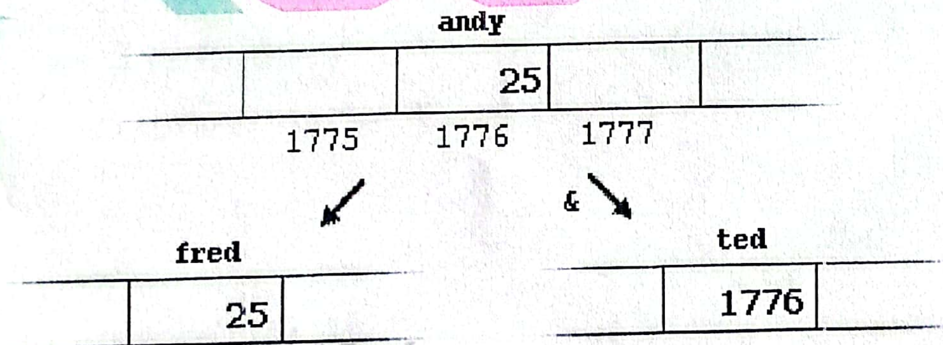
ملاحظة: كلمة العنوان (Address) تكافئ كلمة المرجع (reference) تكافئ الرمز &

مثال: بفرض أنه تم حجز الموقع الذاكري 1776 للمتحول andy أثناء تنفيذ البرنامج، ثم تم تنفيذ التعليمات التالية:

```
andy = 25;
fred = andy;
ted = &andy;
```

والمطلوب: ارسم شكل المواقع الذاكرية ومحتوياتها بعد تنفيذ التعليمات السابقة

الحل:



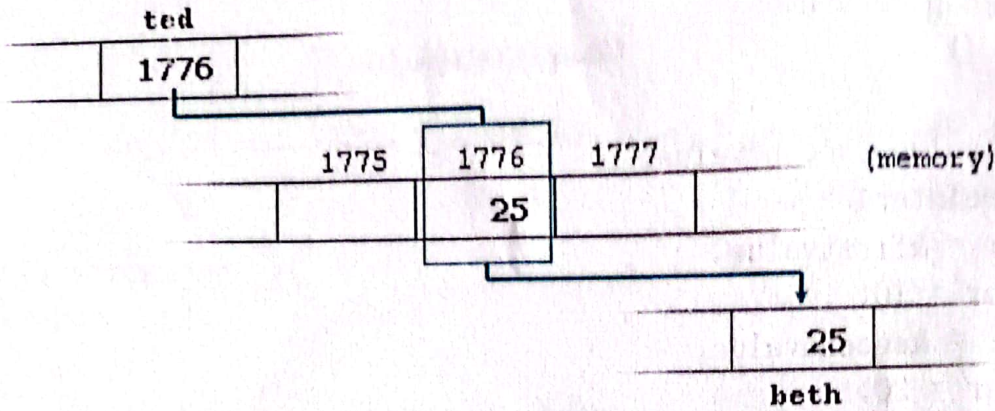
سؤال: أي من المتحولات السابقة هو مؤشر؟

هو ted

ملاحظة: للحصول على القيمة التي يشير لها مؤشر نسبق المؤشر بالرمز *

```
beth = *ted;
```

والشكل التالي يوضح عملية استخدام الرمز *



ملاحظة: الفرق بين معنى الرمز & و الرمز * هو:

- &: يقرأ كالتالي: عنوان المتحول....
- *: يقرأ كالتالي: القيمة المشار لها بالمؤشر....

مثال: بفرض أنه تم حجز الموقع الذاكري 1776 للمتحول andy أثناء تنفيذ البرنامج ، ثم تم تنفيذ التعليمات التالية:

```
andy = 25;
ted = &andy;
```

والمطلوب أوجد قيم:

```
andy
&andy
ted
*ted
```

الحل:

```
andy = 25
&andy = 1776
ted = 1776
*ted = 25
```

التصريح عن المؤشرات

```
int * number;
char * character;
float * greatnumber;
```

مثال: أوجد خرج البرنامج التالي

```
#include <iostream>
using namespace std;
int main ()
{
int firstvalue, secondvalue;
int * mypointer;
mypointer = &firstvalue;
*mypointer = 10;
mypointer = &secondvalue;
*mypointer = 20;
cout << "firstvalue is " << firstvalue << endl;
cout << "secondvalue is " << secondvalue << endl;
return 0;
}
```

الحل: الخرج هو:

```
firstvalue is 10
secondvalue is 20
```

مثال: أوجد خرج البرنامج التالي:

```
#include <iostream>
using namespace std;
int main ()
{
int firstvalue = 5, secondvalue = 15;
int * p1, * p2;
p1 = &firstvalue;
p2 = &secondvalue;
*p1 = 10;
*p2 = *p1;
p1 = p2;
*p1 = 20;
cout << "firstvalue is " << firstvalue << endl;
cout << "secondvalue is " << secondvalue << endl;
system ("pause");
return 0;
}
```

الخرج هو:

```
firstvalue is 10
secondvalue is 20
```

المصفوفات والمؤشرات

سيوضح المثال التالي كيفية الوصول إلى عناصر المصفوفة بطرق مختلفة باستخدام المؤشرات.

مثال: أوجد خرج البرنامج التالي:

```
#include <iostream>
using namespace std;
int main ()
{
int numbers[5];
int * p;
p = numbers; *p = 10;
p++; *p = 20;
p = &numbers[2]; *p = 30;
p = numbers + 3; *p = 40;
p = numbers; *(p+4) = 50;
for (int n=0; n<5; n++)
cout << numbers[n] << ", ";
return 0;
}
```

الحل: الخرج هو

```
10, 20, 30, 40, 50,
```

بناء لائحة مترابطة Linked List

نحتاج في الخوارزميات إلى استخدام تقنيات برمجية وأنواع معطيات كاللائحة المترابطة (Linked List) والمكدس (Stack) والرتل (Queue).

يمكن برمجة هذه الأنواع من الصفر، أو الاعتماد على قوالب مكتبات جاهزة (Standard Template Library: STL) تقدمها لغة C++ للتسهيل والمرونة.

سنبدأ أولاً ببرمجة لائحة مترابطة من الصفر.

ملاحظة: تعتمد لغة C++ على المؤشرات في مفاهيم اللائحة المترابطة.

ملاحظة: للوصول إلى أعضاء الصف (متحولات – توابع) باستخدام الكائن نكتب:

Objet name. Member name

ملاحظة: للوصول إلى أعضاء الصف (متحولات – توابع) باستخدام مؤشر .على الكائن نكتب:

Pointer name-> Member name

مثال: اكتب برنامج يقوم بانشاء لائحة مترابطة مكونة من 3 عقد. تحوي العقدة الأولى على الرقم 1، وتحوي العقدة الثانية على الرقم 2 ، وتحوي العقدة 3 على الرقم 3.

```
// A simple C++ program for
// traversal of a linked list

#include <iostream>
using namespace std;

class Node {
public:
	int data;
	Node* next;
};

// This function prints contents of linked list
// starting from the given node
void printList(Node* n)
{
	while (n != NULL) {
		cout << n->data << " ";
		n = n->next;
	}
}

// Driver's code
int main()
{
	Node* head = NULL;
	Node* second = NULL;
	Node* third = NULL;

	// allocate 3 nodes in the heap
	head = new Node();
	second = new Node();
	third = new Node();
```

```
    head->data = 1; // assign data in first node
    head->next = second; // Link first node with second

    second->data = 2; // assign data to second node
    second->next = third;

    third->data = 3; // assign data to third node
    third->next = NULL;

    // Function call
    printList(head);

    return 0;
}
```



عمليات الإضافة على اللائحة المترابطة Linked List

نعلم أن البنية struck تشبه الصفوف Classes في لغة C++ من حيث التصريح والتعامل معها، لذا:

ملاحظة1: للوصول إلى أعضاء البنية struck (متحولات – توابع) باستخدام الكائن (السجل) نكتب:

Objet name. Member name

ملاحظة2: للوصول إلى أعضاء البنية struck (متحولات – توابع) باستخدام مؤشر على الكائن (السجل) نكتب:

Pointer name-> Member name

1- إضافة عقدة في بداية قائمة احادية الترابط:

مثال: اكتب برنامج بلغة C++ يقوم بإدخال القيم 10 ثم 15 ثم 20 ثم 25 إلى بداية قائمة مترابطة.

```
#include <iostream>
using namespace std;
struct node
{
	int info;
	node *next;
};

class linkedList
{
	private:
	node *first, *last;
	int length;
	public:
	linkedList()
	{
	first = last = NULL;
	length = 0;
	}
	void insertFirst(int item)
	{
	node *newNode = new node;
	newNode->info = item;
	if (length == 0) {
		first = last = newNode;
		newNode->next = NULL;
	}
```

```
        else
        {
                newNode->next = first;
                first = newNode;
        }
        length++;
        }

void print()
{
        node *current = first;
        while (current != NULL)
        {
                cout << current->info << endl;
                current = current->next;
        }
}
};

int main()
{
        linkedList l1;
        l1.insertFirst( 10);
        l1.insertFirst( 15);
        l1.insertFirst( 20);
        l1.insertFirst( 25);
        l1.print();
        system ("pause");
        return 0;
}
```

الخرج هو :

```
25
20
15
10
```

2- إضافة عقدة في نهاية قائمة احادية الترابط:

مثال: أكتب برنامج بلغة ++C يقوم بإدخال القيم 10 ثم 15 ثم 20 ثم 25 إلى نهاية قائمة مترابطة

```
#include <iostream>
using namespace std;
struct node
{
        int info;
```

```
    node *next;
};

class linkedList
{
    private:
    node *first, *last;
    int length;
    public:
    linkedList()
    {
    first = last = NULL;
    length = 0;
    }

    void insertLast (int item)
    {
    node* newNode = new noce;
    newNode->info = item;
    if (length==0)  {
        newNode->next =NULL;
        first =last=newNode;
    }
    else
    {
    last ->next= newNode;
    newNode->next=NULL;
    last=newNode;
    }
    length++;
    }

void print()
{
    node *current = first;
    while (current != NULL)
    {
        cout << current->info << endl;
        current = current->next;
    }
}
};

int main()
{
    linkedList l1;
    l1.insertLast( 10);
    l1.insertLast( 15);
```

```
    l1.insertLast( 20);
    l1.insertLast( 25);
    l1.print();
    system ("pause");
    return 0;
}
```

الخرج هو:

```
10
15
20
25
```

3- إضافة عقدة في موقع محدد في قائمة احادية الترابط

ينفذ التابع insertAt ادخال عقدة بعد موضع محدد pos، وعندما يكون pos=0 يؤول إلى الإضافة من المقدمة حيث يتم استدعاء insertFirst وعندما يكون pos=length يؤول إلى الإضافة من النهاية حيث يتم استدعاء insertLast.

```
void insertAt(int pos, int item)
{
    node *newNode = new node;
    newNode->info = item;
    if (pos == 0) //insert at the begining
        insertFirst(item);
    else if (pos == length) //insert at the end;
        insertLast(item);
    else
    {
        node *current = first;
        for (int i = 1; i < pos; i++)
        current = current->next;
        newNode->next = current->next;
        current->next = newNode;
        length++;
    }
}
```



تمارين على المكدس والرتل

نحتاج في الخوارزميات إلى استخدام تقنيات برمجية وأنواع معطيات كالمكدس Stack والرتل Queue واللائحة List.

يمكن برمجة هذه التقنيات من الصفر، أو الاعتماد على قوالب مكتبات جاهزة (Standard Template Library: STL) تقدمها لغة C++ للتسهيل والمرونة.

في الأمثلة الثلاثة التالية اعتمدنا على تقديم هذه التقنيات بواسطة المكتبات الجاهزة: #include <stack> و #include <queue> و #include <list>

مثال على طريقة عمل المكدس بواسطة مكتبة جاهزة هي #include <stack>

سنستخدم التوابع الجاهزة التالية:

push() يقوم بإدخال ودفع قيمة للمكدس

pop() يقوم بإخراج وسحب قيمة من المكدس

empty() يفحص فيما إذا كان المكدس فارغاً أم لا

top() يعيد القيمة الموجودة في رأس المكدس

size() يعيد عدد عناصر المكدس

```
#include <iostream>
#include <stack>
using namespace std;
void showstack(stack <int> s)
{
      while (!s.empty())
      {
            cout << '\t' << s.top();
            s.pop();
      }
      cout << '\n';
}
int main ()
{
      stack <int> s;
```

```
        s.push(10);
        s.push(30);
        s.push(20);
        s.push(5);
        s.push(1);
        cout << "The stack is : ";
        showstack(s);
        cout << "\ns.size() : " << s.size();
        cout << "\ns.top() : " << s.top();
        cout << "\ns.pop() : ";
        s.pop();
        showstack(s);
        system ("pause");
        return 0;
}
```

الخرج

```
The stack is :     1     5     20     30     10

s.size() : 5
s.top() : 1
s.pop() :     5     20     30     10
```

مثال عن طريقة عمل الرتل بواسطة مكتبة جاهزة هي `#include <queue>`

سنستخدم التوابع الجاهزة التالية:

push() يقوم بإدخال قيمة للرتل

pop() يقوم بإخراج وسحب قيمة من الرتل

empty() يفحص فيما إذا كان الرتل فارغاً أم لا

front() يعيد القيمة الموجودة في أول الرتل

back() يعيد القيمة الموجودة في آخر الرتل

size() يعيد عدد عناصر الرتل

```
#include <iostream>
#include <queue>
using namespace std;
void showq(queue <int> gq)
{
    while (!gq.empty())
    {
        cout << '\t' << gq.front();
        gq.pop();
    }
```

```
    cout << '\n';
}

int main()
{
    queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);

    cout << "The queue gquiz is : ";
    showq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);
     system("pause");
    return 0;
}
```

الخرج:

```
The queue gquiz is :     10     20     30

gquiz.size() : 3
gquiz.front() : 10
gquiz.back() : 30
gquiz.pop() :      20     30
```